Oracle relationnel objet

présnetation

- La norme SQL 3 aussi appelée SQL'99, a introduit les extensions objet au modèle relationnel.
- Fonctionnalités objet dans Oracle
  - Types définis par l'utilisateur et le type object
  - Collections
  - Tables imbriquées
  - OID: implémentation
  - Les relations : ref
  - Cluster
  - Héritage
  - Encapsulation

1- Le type object et les types définis par l'utilisateur.

- Création d'un type

```
create or replace type
TPersonne as object(
id int,
nom varchar2(80))
```

- Création d'une collection

```
create or replace type
TPersonnes as varray(3) of
TPersonne
```

- Utilisation d'un type objet

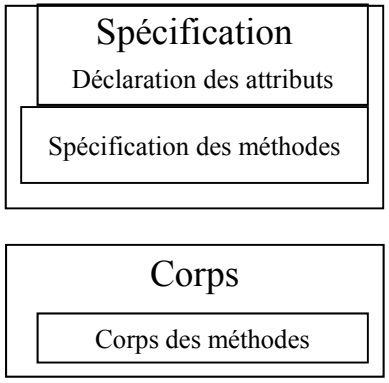
```
CREATE TABLE cours
   (idcoursNUMBER(*,0),
titre VARCHAR2(20 BYTE),
enseignant TPERSONNE ,
Etudiants TPERSONNES
   )
```

- Suppression d'un type : `Drop type Tpersonne Force;`

  (force est obligatoire si des objets ont déjà été créés)

Les objets

- Un objet est constitué par :
  - une spécification qui détermine l'interface publique du type, qui contient
    - une structure de données (déclaration d'attributs) → état
    - la spécification des méthodes permettant leur manipulation → comportement
      - CREATE TYPE <nom> AS OBJECT <spécification>
  - une implémentation (corps), qui contient
    - l'implémentation des méthodes définies par la spécification
      - CREATE TYPE BODY AS <implémentation>



Interface publique

Implémentation privée

- Les méthodes
  - Méthodes membres
  - Méthodes statiques
  - Constructeurs (un constructeur par défaut est défini pour chaque type).
- Méthodes statiques
  - Déclaration static function ou static procedure (et aussi dans l'implémentation, il faut utiliser le mot clé static)
  - Le mot clé self n'est pas passé en paramètre.
- Constructeurs
  - Pour appeler le constructeur le mot clé new n'est pas obligatoire.

Exemple : création du type TBoite

```
create or replace TYPE TBoite AS OBJECT (
lo INTEGER,
la INTEGER,
ha INTEGER,
MEMBER FUNCTION surface RETURN INTEGER,
MEMBER FUNCTION volume RETURN INTEGER,
MEMBER PROCEDURE afficher ();
/
```

Implémentation du type TBoite

```
create or replace TYPE BODY TBoite  AS
  MEMBER FUNCTION volume RETURN INTEGER IS
  BEGIN
  RETURN la * ha * lo;
  -- ou bien RETURN SELF.la * SELF.ha * SELF.lo; --
  END;
  MEMBER FUNCTION surface RETURN INTEGER IS
  BEGIN --
  RETURN 2 * (lo * la + lo * ha + la * ha);
  END;
  MEMBER PROCEDURE afficher (SELF IN OUT NOCOPY Tboite) IS
  BEGIN
  DBMS_OUTPUT.PUT_LINE('Longueur: ' || lo || ' - ' || 'Largeur: ' ||
  la
  || ' - ' || 'Hauteur: ' || ha);
  DBMS_OUTPUT.PUT_LINE('Volume: ' || volume || ' - ' || 'Surface: '
  || surface);
  END;

END;
```

Utilisation du type TBoite

- Création d'une table de type TBoite

```
create table boite of TBoite;
```

- Ajout d'une ligne dans la table boite

```
INSERT INTO boite VALUES(10, 10, 10);
-- ou bien
INSERT INTO boite
VALUES(TBoite(10, 10,10));
```

- Sélection

```
-- Sélection de toutes les lignes
SELECT * FROM boite;
-- Sélection du volume et de la surface des boîtes dont lo=10
SELECT s.volume(), s.surface() FROM tboite s WHERE s.lo = 10;
```

- Utilisation dans un bloc PL/SQL

```
DECLARE
b TBoite;
BEGIN
-- affectation du résultat d'une instruction SQL dans une
variable
SELECT VALUE(s) INTO b FROM boite s WHERE s.lo = 10;
b.afficher();
END;
```

Exemple 2: le type TPersonne

Spécification du type TPersonne

```
create type TPersonne as
Object(
idN number,
prenom varchar2(20),
nom varchar2(20),
email varchar2(20),
tel varchar2(20),
member function getId return
number,
member procedure
afficherDetails);
```

Implémentation du type TPersonne

```
create type body TPersonne as
    map member function getId
    return number is
    begin
    return idN;
    end;
    member procedure
    afficherDetails is
    Begin
    dbms_output.put_line
    (to_char(idN) || ' ' ||
    prenom || ' ' || nom);
    dbms_output.put_line(email
    || ' ' || tel);
    end;
end;
```

- Création d'une table contacts contenant un champ de type TPersonne

```
create table contacts (
contact TPersonne,
contact_date Date);
```

- Sélection des id de tous les enregistrement de la table contacts

```
Select c.contact.getId()
from contacts c;
```

- Ajout d'une ligne dans la table contacts

```
insert into contacts values
(TPersonne(25,'nom','prenom','em@emails.com','455555'),'23/1
0/2012');
```

- Remarque: Une table de type objet peut être considérée comme une table constituée de lignes et de colonnes ou une table contenant une seule colonne où chaque ligne contient un objet.
- Chaque objet doit être identifié par un OID unique qui est généré automatiquement par le système (par défaut), ou bien nous pouvons considérer que l'OID est la clé primaire de la table
  - Create table nom_table (…) object identifier is primary key (par défaut object identifier is system generated) . Taille de l'OID système: 16 octets.

Tables imbriquées

```
create or replace TYPE T_Personne AS TABLE OF TPersonne;

CREATE TABLE Cours (
idCours number,
titre VARCHAR2(20),
enseignants T_Personne)
NESTED TABLE enseignants  STORE AS Enseignant;
```

Le mot clé ref

```
create table Enseignants of TPersonne
```

```
create table Cours (
idCours int,
titre varchar2(20),
enseignant ref TPersonne scope is enseignants)
```

- Le mot clé « ref » permet de définir une référence dans un objet `vers` un autre objet, le mot clé « ref » peut remplacer le mot clé « join ».
- L'instruction « scope is » permet de spécifier la table référencée par l'objet (dans ce cas la colonne stockera uniquement l'OID de l'objet référencé), si le paramètre « scope » est omis et que plusieurs tables sont de type TPersonne, alors le moteur de la base de données parcourera un ensemble de références d'objets afin de localiser l'enregistrement.

Relations avec ref

```
select c.titre
from cours c
where c.enseignant.nom='Daudet'
```

cluster

```
create cluster HD_Cluster (
id varchar2(2);
)
```

Héritage : under (à partir de 9i)

```
create or replace
type TPersonne1 as object (
idPersonne int,
nom varchar2(20)) not final
```

```
create type TEtudiant under TPersonne1
(
cours varchar2(20),
annee varchar2(4))
```

```
create table Personne of TPersonne1 (
idPersonne not null,
primary key (idPersonne))
```

Encapsulation

- Deux types d'encapsulation
  - Procédures ou fonctions stockées
  - Procédures ou fonctions membres

Procédures stockées

```
CREATE USER user3 IDENTIFIED BY 111;
  GRANT EXECUTE ON supprimerCours TO user3;
CREATE OR REPLACE
PROCEDURE supprimerCours(
    idC cours.idcours%type)
AS
BEGIN
  DELETE FROM cours WHERE idcours=idC;
END supprimerCours;
EXECUTE supprimerCours (1);
```

Fonction

```
CREATE OR REPLACE
  FUNCTION nomEnseignant(
      id INT)
    RETURN VARCHAR2
  IS
    n VARCHAR2(20);
  BEGIN
    SELECT nom INTO n FROM enseignant WHERE
ippersonne=id;
    RETURN n;
  END nomEnseignant;
```

Procédures ou fonctions membres

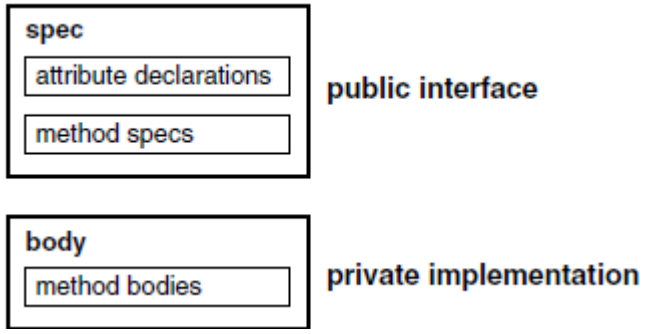
```
CREATE OR REPLACE type TEtudiant1 AS Object
  (
    id    INT,
    cours VARCHAR2(20),
    member PROCEDURE supprimerEtudiant)
CREATE OR REPLACE type Body TEtudiant1
AS
  member PROCEDURE supprimerEtudiant
IS
BEGIN
  DELETE FROM etudiant2 WHERE etudiant2.id = self.id;
END supprimerEtudiant;
END;
CREATE TABLE Etudiant2 OF TEtudiant1
  ( id NOT NULL, PRIMARY KEY (id)
  )
```

```
CREATE TYPE <object1 schema>
(attr1 data type,...,
 attrj data type,

 PROCEDURE <procedure1 name>(
 param1 parameter type data type,..
 paramn parameter type data type);
) NOT FINAL/

CREATE TYPE <object2 schema> UNDER
(attr1 data type,...,
 attrj data type,

 OVERRIDING PROCEDURE <procedure1 r
 param1 parameter type data type,..
 paramn parameter type data type);
)/
```

```
CREATE TYPE Personne2
(idPersonne VARCHAR(10),
prenom VARCHAR(20),
nom VARCHAR(20),
adresse ADDRESS,
dateNaissance DATE,
PRIVATE FUNCTION getTelephone,
PUBLIC PROCEDURE getInfos);

CREATE TYPE Staff UNDER Personne2
(PUBLIC StaffStartDate DATE,
PROTECTED StaffPhone CHAR(10),
PRIVATE StaffPayRate DECIMAL(5,2),
PRIVATE StaffCommRate DECIMAL(5,2),
PUBLIC FUNCTION RetrieveTotalPayment)
```

Méthodes membres

- Méthodes membres
- Méthodes statiques
- Les constructeurs.



2- L'Objet-relationnel dans Oracle

1- Les relations

1-1 L'héritage
1-1-1 type: union (overlapping , complete)

Chaque instance du type parent est au moins du type de l'un des sous types.

Les types

```
CREATE OR REPLACE type
TPersonne
AS Object(
    id      INT,
    nom
VARCHAR2(20),
    adresse
VARCHAR2(20) ) NOT
final;
```

```
CREATE type TEtudiant
under TPersonne
( cours VARCHAR2(20),
annee VARCHAR2(4));
```

```
CREATE type TEnseignant
under TPersonne
( departement VARCHAR2(20),
classe VARCHAR(4));
```

Les tables

```
CREATE TABLE Personne OF
TPersonne
  ( id NOT NULL, PRIMARY
KEY (id)
  );
```

```
CREATE TABLE
Enseignat OF
TEnseignant
  ( id NOT NULL,
PRIMARY KEY (id)
  );/
```

```
CREATE TABLE Etudiant
OF TEtudiant
  ( id NOT NULL,
PRIMARY KEY (id)
  ); /
```

1-1-2 exclusion (disjoint, incomplete)

```
CREATE OR REPLACE
type TEmploye under
TPersonne
(
  emptype VARCHAR2(8)
)
NOT final;
/
```

- Par défaut un type est final, pour les méthodes la valeur par défaut est not final.
- Alter type nomType Final (uniquement si'il n'exite pas déjà des sous types) ou
- Alter type nomType Not Final

```
CREATE type TManager
under TEmploye
( salaire_annuel
NUMBER)
/
```

```
create type TOuvrier
under TEmploye (
salaire_journalier
number)
/
```

```
CREATE TABLE Employe OF
TEmploye
  (id NOT NULL,
    emptype CHECK (emptype
IN ('manager',
'ouvrier','null')),
    PRIMARY KEY (id)
  );
```

1-1-3 partition (disjoint , complete)

Similaire au cas précédent, l'unique différence est que emp_type doit être not null.

emtype not null,

emptype check (emptype in ('Manager', 'Ouvrier')).

Surcharge (Overload) et Redéfinition (override) des méthodes

- Pour redéfinir une méthode Utiliser le mot clé « **OVERRIDING »** devant « member » dans la déclaration et dans la définition de la méthode.
- Dans une méthode d'un type dérivé, l'appel d'une méthode du type parent se fait de la manière suivante:

  (self as Type_Parent).nom_methode

Types et méthodes abstraites

- Type: create type …) NOT INSTANTIABLE not final
- Méthode: NOT INSTANTIABLE MEMBER FUNCTION

1-2 Les associations

Une association peut être implémentée de deux manières différentes:

- Clés primaires et clés étrangères
- Références vers des objets: l'attribut qui connecte les deux tables contient une référence vers l'emplacement où est stockée la table connectée.

1-2-1 n-n

1re méthode:

```
CREATE TABLE cours1
  ( idcours INT,
titre VARCHAR2(10),
PRIMARY KEY (idcours)
  );
/
```

```
CREATE TABLE
etudiant1
  ( idetu INT, nom
VARCHAR2(20), PRIMARY
KEY (idetu)
  );
/
```

```
CREATE TABLE cours_etu
  (
    courid INT NOT NULL,
    etuid  INT NOT NULL,
    PRIMARY KEY (courid,etuid),
    FOREIGN KEY (courid)
REFERENCES cours1 (idcours) ON
  DELETE CASCADE, FOREIGN KEY
(etuid) REFERENCES etudiant1
(idetu) ON
  DELETE CASCADE
  );/
```

2e méthode:

```
CREATE TABLE Etudiant
OF TEtudiant
  ( id NOT NULL,
PRIMARY KEY (id)
  );
/
```

```
CREATE TABLE Cours OF TCours
  ( idCours NOT NULL, PRIMARY
KEY (idCours)
  );
/
```

```
CREATE TABLE cours_etudiants
  ( idCours ref TCours,
    idEtudiant ref TEtudiant
  );
/
```

1-2-2 1-n

1re méthode:

```
CREATE TABLE
Formateur
  ( idfo INT, nom
VARCHAR2(20), PRIMARY
KEY (idFo)
  );
/
```

2e méthode:

```
CREATE type TCours2
AS Object
  ( idCours INT,
    titre
VARCHAR2(20),
    formateur ref
TEnseignant
  );/
```

```
CREATE TABLE cours
  (idcours INT,
    titre   VARCHAR2(10),
    PRIMARY KEY (idcours),
    idFo INT ,
    FOREIGN KEY (idFo)
REFERENCES Formateur(idFo) ON
  DELETE CASCADE
  );
/
```

```
CREATE TABLE cours2
OF TCours2    (
idcours INT NOT NULL,
PRIMARY KEY (idcours)
    ); /
```

```
CREATE TABLE
Formateur2 OF
TEnseignant
    ( id NOT NULL,
PRIMARY KEY (id)
    );/
```

1-2-3 1-1

1re méthode:

- Contrainte de participation: la participation de formateur dans l'association est totale, la participation de bureau est partielle: la clé primaire de la table avec une participation partielle devient une clé étrangère de la table avec une participation totale

```
 CREATE TABLE formateur3
    (idFo INT NOT NULL,
nom  VARCHAR2(20),
PRIMARY KEY (idFo),
idBureau INT ,
FOREIGN KEY (idBureau)
REFERENCES
Bureau(idbureau)
    );
  /
```

```
 CREATE TABLE bureau
     idbureau INT NOT
NULL,
Batiment
VARCHAR2(10),
PRIMARY KEY
(idbureau)
    );  /
```

2e méthode:

```
CREATE TYPE TBureau
AS
  OBJECT
  (
    idbureau INT,
    batiment VARCHAR2(20)
  )
  /
```

```
CREATE TABLE Bureau OF TBureau
    ( idBureau NOT NULL,
PRIMARY KEY (idBureau)
    );
  /
```

```
CREATE TYPE TPersonne2
AS
  OBJECT
  (id  INT,
    nom VARCHAR2(30),
   bureau REF TBureau
  ) /
```

```
 CREATE TABLE Formateur5 OF
TPersonne2
    ( id NOT NULL, PRIMARY KEY
(id)
    );
  /
```

1-3 l'agrégation

- 4 types d'agrégation
  - Une relation d'agrégation peut être partagée ou non.
  - Dépendante ou non dépendante: l'existence d'un composant peut dépendre de l'existence de son composite (la suppression de l'agrégat entraine la suppression de tous les objets agrégés) ou non.
  - Une agrégation peut être homogène ou bien hétérogène

- Deux techniques d'implémentation
  - Clustering: Un cluster est un objet de schéma qui contient des données provenant d'une ou de plusieurs tables qui ont une ou plusieurs colonnes en commun. La base de données Oracle stocke ensemble toutes les lignes tables qui partagent la même clé de cluster.
  - Imbrication

- Exemple une agrégation dépendante et partagée.

```
CREATE cluster
HDCluster (ihd INT)
```

```
 CREATE TABLE hd_Contr
    (
      ihd          INT NOT NULL,
      ihdc         INT NOT NULL,
      description VARCHAR2(20),
      PRIMARY KEY (ihd,ihdc),
      FOREIGN KEY (ihd) REFERENCES
HD(ID_HD)
    )
    cluster HDCluster
    (
      ihd
    );
  CREATE INDEX cluster_hd_index ON
cluster HDCluster ;
```

```
 CREATE TABLE HD
    (
      id_Hd    INT
NOT NULL,
      capacite
VARCHAR2(10),
      PRIMARY KEY
(id_Hd)
    )
    cluster HDCluster
    (
      id_Hd
    );
```

- Agrégation dépendante et non partagée

```
CREATE TABLE HD_Contr
(hd_id VARCHAR2(10) NOT NULL,
hd_contr_id VARCHAR2(10) NOT NULL,
description VARCHAR2(25),
PRIMARY KEY (hd_contr_id),
FOREIGN KEY (hd_id) REFERENCES Hard_Disk (hd_id))
CLUSTER HD_Cluster(hd_id);
```

```
CREATE TABLE Hard_Disk
(hd_id VARCHAR2(10) NOT NULL,
capacity VARCHAR2(20),
PRIMARY KEY (hd_id))
CLUSTER HD_Cluster(hd_id);
```

- Agrégation dépendante non partagée: utilisation des tables imbriquées

```
CREATE type HDControleur
AS
  TABLE OF THDControleur
```

```
CREATE type THDControleur
AS
  Object
  (
    idHdc         INT,
    desccription
VARCHAR2(10));
```

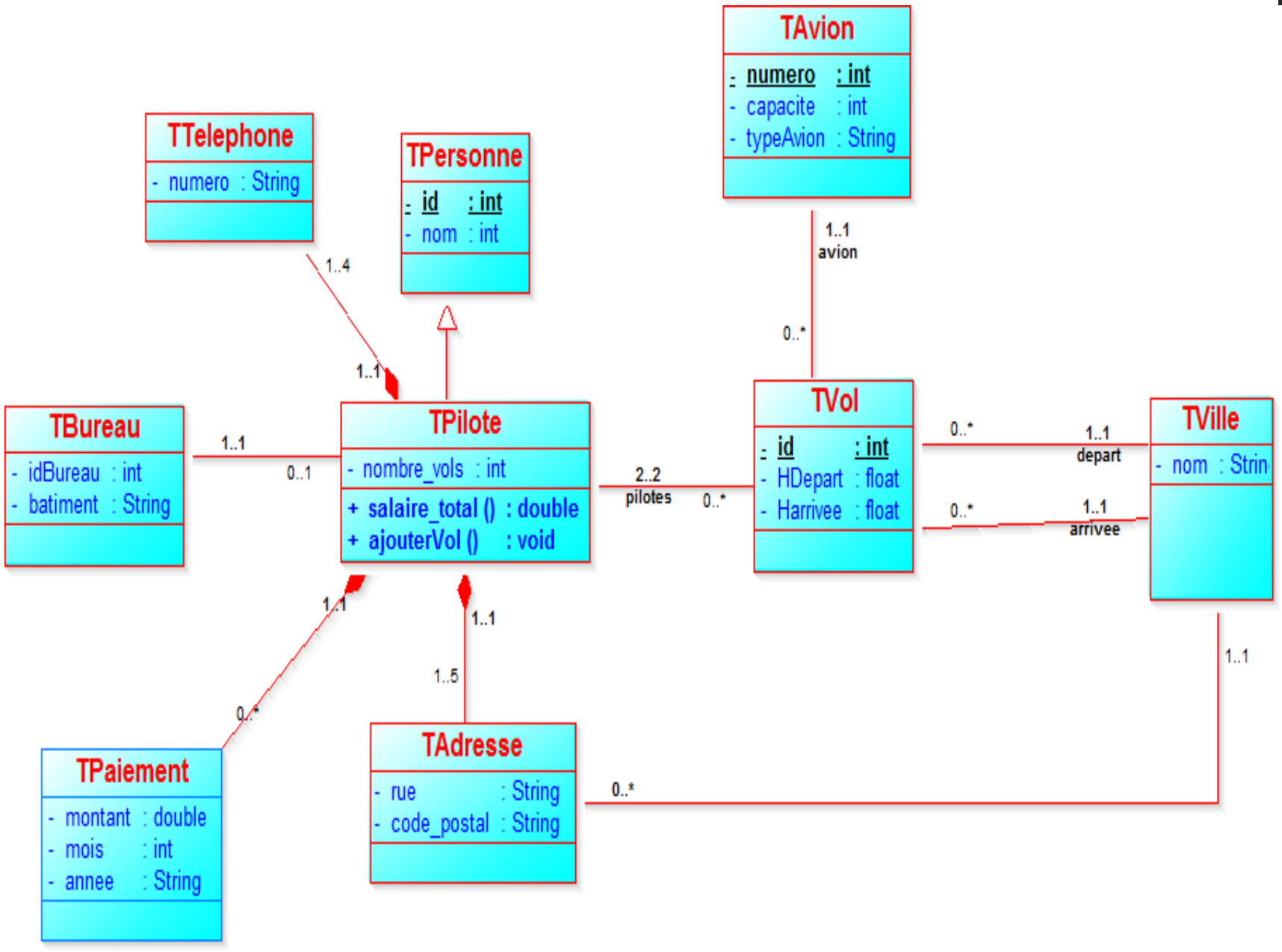
```
CREATE TABLE Disque
    (
      hd_id    VARCHAR2(10) NOT NULL,
      capacit2 VARCHAR2(20),
      controleur HDControleur,
      PRIMARY KEY (hd_id)
    )
    NESTED TABLE controleur STORE AS HD_Contr_tab;
```

Utilisation d'une table imbriquée

```
INSERT
  INTO disque VALUES
    (
      '1',
      '500 Go',
      HDControleur( THDControleur(11,'Un'),
THDControleur(22,'Deux'))
    );
```

```
 SELECT g.* FROM disque d, TABLE(d.controleur) g;
```

Exemple


TAvion
- numero : int
- capacite : int
- typeAvion : String
TTelephone
- numero : String
TPersonne
- id : int
- nom : int
1..4
1..1
avion
0..*
TBureau
- idBureau : int
- batiment : String
1..1
0..1
TPilote
- nombre_vols : int
+ salaire_total () : double
+ ajouterVol () : void
2..2
pilotes
0..*
TVol
- id : int
- HDepart : float
- Harrivee : float
0..*
1..1
depart
0..*
1..1
arrivee
TVille
- nom : Strin
1..1
1..1
1..5
0..*
TPaiement
- montant : double
- mois : int
- annee : String
TAdresse
- rue : String
- code_postal : String
0..*

Choix entre varray et table imbriquée

- Différence entre varray et table imbriquée :
  1. varray est une collection ordonnée
  2. La taille d'une collection varray est limitée, varray utilise le support de stockage d'une manière plus optimisée qu'une table imbriquée.
  3. Nous pouvons interroger une table imbriquée, mais pas une collection varray.
  4. Les lignes d'une table peuvent être indexées, mais une collection varray.

- Si aucun des deux premiers points n'est important alors
  - Si vous avez besoin d'interroger les éléments de la collection → table imbriquée.
  - Sinon varray (la collection sera récupérée comme un ensemble.)

Le package UTL_REF

- Le package utl_ref contient les procédures pour supporter les opérations basées sur les références.
  - Select_Object retourne l'objet référencé
    - Utl_ref.Select_Object(refO , o): affecte l'objet référencé par refO dans o.
  - Delete_Object
  - Update_Object
  - Lock_Object

Utilisation des types ref

- Création:

```
create or replace
type TPilote under
TPersonne (
nombre_vols int,
bureauP ref TBureau --
association de type 1-1
);
```

```
create or replace
type TPersonne as object (
id int,
nom varchar2(20)) not final;
```

```
create table Pilote of TPilote (
id not null,
primary key (id),
bureauP not null,
foreign key (bureauP) references Bureau )
;
```

- Insertion

```
declare
bur ref TBureau;
begin
select ref(b) into bur from bureau b where
b.idbureau=1;
insert into pilote2 values (1,'p1',0,bur);
end;
```

- Modification

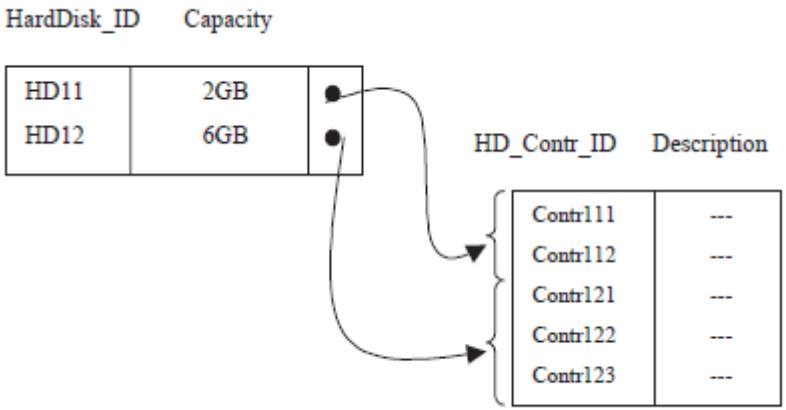
```
declare
bur ref TBureau;
begin
select ref(b) into bur from bureau b where
b.idbureau=1;
update pilote2 p set p.bureauP=bur where p.id=2;
end;
```

- Sélection

```
select
p.id,p.nom,p.nombre_vols,deref(p.bureaup).idBureau,
deref(p.bur
eaup).batiment from pilote2 p;
```

Clusters et tables imbriquées

hd id	capacity	hd contr id	description
HD11	2GB	Contr111	.....
		Contr112	.....
HD12	6GB	Contr121	.....
		Contr122	.....
		Contr123	.....



Agrégation dépendante

- Tables imbriquées (supporte seulement les agrégations partagées)