Conception de bases de données

Tables d'objets en relationnel-objet sous Oracle

roo2.pdf

utc Formation

Stéphane Crozat

11 mai 2017

Table des matières

Cours

I

A. Tables d'objets et OID en SQL3

Objectifs

Connaître l'extension du LDD pour déclarer des tables d'objets
Connaître l'extension du LMD pour naviguer des tables d'objets

1. Création de table objet (extension au LDD SQL)

Rappel

Création de type en SQL3 sous Oracle (extension au LDD) - p.27

Syntaxe : LDD SQL3

```
1  CREATE TABLE nom_table OF nom_type (
2    PRIMARY KEY(attribut1),
3    attribut2 NOT NULL,
4    UNIQUE (attribut3)
5    FOREIGN KEY (attribut4) REFERENCES ...
6  );
```

Il est possible, sur une table ainsi définie, de spécifier les mêmes contraintes que pour une table créée avec une clause CREATE TABLE (contraintes de table). Ces contraintes doivent être spécifiées au moment de la création de la table, et non au moment de la création du type (bien que la définition de type permet de spécifier certaines contraintes, comme NOT NULL).

Exemple : Héritage

```
1  CREATE OR REPLACE TYPE typIntervenant AS OBJECT(
2  pknom varchar2(20),
3  prenom varchar2(20)
4  );
5  /
6
7  CREATE TABLE tIntervenant OF typIntervenant (
8  PRIMARY KEY(pknom),
9  prenom NOT NULL
10 );
```

2. Références entre enregistrements avec les OID (extension au LDD SQL)

Syntaxe : LDD SQL3

```
CREATE TYPE  type_objet AS OBJECT ...

CREATE TABLE table1 OF type_objet ...

CREATE TABLE table2 (
  ...
  clé_étrangère REF type_objet,
  SCOPE FOR (clé_étrangère) IS table1,
  );
```

Remarque : SCOPE FOR

Renforce l'intégrité référentielle en limitant la portée de la référence à une table particulière (alors que REF seul permet de pointer n'importe quel objet de ce type)

Exemple

```
CREATE OR REPLACE TYPE typCours AS OBJECT(
pkannee number(4),
pknum number(2),
titre varchar2(50),
type char(2),
refintervenant REF typIntervenant,
debut date,
MEMBER FUNCTION fin RETURN date
);
/

CREATE TABLE tCours OF typCours (
CHECK (pkannee>2000 and pkannee<2100),
type NOT NULL,
CHECK (type='C' or type='TD' or type='TP'),
refintervenant NOT NULL,
SCOPE FOR (refintervenant) IS tIntervenant,
PRIMARY KEY(pkannee, pknum)
);
```

3. Insertion de références par OID (INSERT)

Pour faire une référence avec un OID, il est nécessaire de récupérer l'OID correspondant à l'enregistrement que l'on veut référencer.

L'OID est accessible grâce à la syntaxe REF en SQL3.

Sous Orcale, on utilisera une procédure PL/SQL pour récupérer l'OID dans une variable avant de pouvoir l'insérer.

Syntaxe : REF

```
SELECT REF(alias)
FROM nom_table alias
```

```

0000280209DB703686EF7044A49F8FA67530383B36853DE7106BC74B6781275ABE5A553A5F01C0003
40000
```

Syntaxe : Insertion de données en PL/SQL

```
DECLARE
  variable REF type_objet;
BEGIN
  SELECT REF(alias) INTO variable
  FROM table2 alias
  WHERE clé_table2='valeur';

  INSERT INTO tCours (champs1, ..., clé_étrangère)
  VALUES ('valeur1', ...,  variable);
END;
```

Exemple

```
DECLARE
 refI REF typIntervenant;
BEGIN
 INSERT INTO tIntervenant (pknom, prenom)
 VALUES ('CROZAT', 'Stéphane');

 SELECT REF(i) INTO refI
 FROM tIntervenant i
 WHERE pknom='CROZAT';

 INSERT INTO tCours (pkannee, pknum, titre, type, debut, refintervenant)
 VALUES ('2003', 1, 'Introduction','C', '01-JAN-2001', refI);

 INSERT INTO tCours (pkannee, pknum, titre, type, debut, refintervenant)
 VALUES ('2003', 2, 'Modélisation','TD', '02-JAN-2001', refI);
END;
/
```

4. Manipulation d'OID par la navigation d'objets (SELECT)

Syntaxe : Navigation d'objets

La référence par OID permet la navigation des objets sans effectuer de jointure, grâce à la syntaxe suivante :

```
SELECT t.reference_oid.attribut_table2
FROM table1 t;
```

Exemple

```
SELECT c.pkannee, c.pknum, c.refintervenant.pknom
FROM tCours c
```

```
PKANNEE PKNUM REFINTERVENANT.PKNOM
------- ----- --------------------
   2003     1 CROZAT
   2003     2 CROZAT
```

5. RO sans fil référencé

[30 minutes]

L'on souhaite réaliser une base de données permettant de gérer tous les relais Wifi sur un site d'entreprise. Chaque relais est identifié par une coordonnée géographique (représentée par une chaîne de caractère) et fait référence à un modèle. On associe également à chaque relais

son prix d'achat. Chaque modèle est décrit par une marque et un type et possède une puissance.

Question 1

[Solution n°1 p 17]

Proposez une implémentation RO SQL3 sous Oracle exploitant les tables objets. Vous proposerez un MCD et un MLD préalablement pour vous aider.

Question 2

[Solution n°2 p 18]

Insérer les données suivantes dans votre base de données :

- Un modèle de marque SuperWif et de type X1, puissance 25mW
- Deux relais de ce modèle respectivement aux coordonnées (48.853,2.35) et (48.978,3.01), achetés chacun 100€.

Question 3

[Solution n°3 p 18]

Écrivez deux requêtes permettant de renvoyer respectivement :

- La puissance du relais 1
- La moyenne des prix des relais pour chaque modèle

B. Compléments

1. Méthodes dans les tables d'objets sous Oracle

a) Méthodes de table d'objets

Définition : Méthodes de table

Si le type sur lequel s'appuie la création de la table définit des méthodes, alors les méthodes seront associées à la table (méthodes de table).

Il sera possible d'accéder à ces méthodes de la même façon que l'on accède aux attributs (projection, sélection...).

Syntaxe : Accès aux méthodes d'une table d'objets

```
1  SELECT t.m1(), t.m2() ...
2  FROM table t
3  ...
```

Attention

L'utilisation d'un alias est obligatoire pour accéder aux méthodes.

Syntaxe : Déclaration de type avec méthodes

```
1  CREATE TYPE nom_type AS OBJECT (
2   nom_attribut1 type_attribut1
3   ...
4   MEMBER FUNCTION nom_fonction1 (parametre1 IN|OUT type_parametre1, ...) RETURN
    type_fonction1
5   ...
6  );
7  /
```

```
CREATE TYPE BODY nom_type
 IS
 MEMBER FUNCTION nom_fonction1 (...) RETURN type_fonction1
  IS
  BEGIN
  ...
  END ;
 MEMBER FUNCTION nom_fonction2 ...
  ...
 END ;
END ;
/
```

Exemple

```
CREATE TYPE typCours AS OBJECT (
  pknum NUMBER(2),
  debut DATE,
  MEMBER FUNCTION fin RETURN DATE
);
/
CREATE TYPE BODY typCours IS
MEMBER FUNCTION fin RETURN DATE
  IS
  BEGIN
    RETURN SELF.debut + 5;
  END;
END;
/

CREATE TABLE tCours OF typCours (
pknum PRIMARY KEY
);

SELECT c.pknum, c.fin()
FROM tCours c;
```

Remarque : Type retourné par une méthode

« The datatype cannot specify a length, precision, or scale. »
http://docs.oracle.com/cd/B13789_01/server.101/b10759/statements_5009.htm[1]

b) Méthodes et SELF

SELF

Lorsque l'on écrit une méthode on a généralement besoin d'utiliser les attributs propres (voire d'ailleurs les autres méthodes), de l'objet particulier que l'on est en train de manipuler.
On utilise pour cela la syntaxe SELF qui permet de faire référence à l'objet en cours.

Syntaxe : SELF

```
self.nom_attribut
self.nom_méthode(...)
```

Exemple : Total d'une facture

```
MEMBER FUNCTION total RETURN number
IS
  t number;
BEGIN
```

1 - http://docs.oracle.com/cd/B13789_01/server.101/b10759/statements_5009.htm

```
5     SELECT sum(f.qte) INTO t
6     FROM facture f
7     WHERE f.num=self.num;
8
9    RETURN t;
10  END total;
```

Remarque : SELF implicite

Dans certains cas simples, lorsqu'il n'y a aucune confusion possible, SELF peut être ignoré et le nom de l'attribut ou de la méthode directement utilisé.

Il est toutefois plus systématique et plus clair de mettre explicitement le self.

Exemple : Exemple de SELF implicite

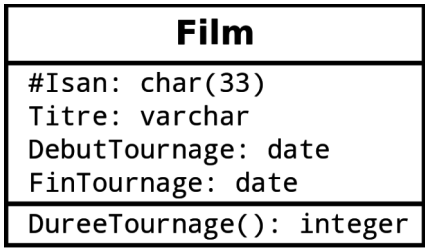
```
1   MEMBER FUNCTION adresse RETURN varchar2
2   IS
3   BEGIN
4     RETURN num || rue || ville;
5   END;
```

c) Un film vraiment méthodique

Question

[*Solution n°4 p 18*]

Proposer une implémentation SQL3 sous Oracle correspondant au modèle UML. La méthode `DureeTournage` sera implémentée de façon à retourner un nombre entier de jours.



Film (méthode)

Indices :

On pourra utiliser une fonction `duree(debut,fin)` qui retourne le nombre de jours entre deux dates.

On fera un usage explicite de SELF.

2. Compléments SQL3 pour les tables objets

a) Insertion d'OID sans mobiliser de PL/SQL (INSERT SELECT)

Syntaxe : Insertion de référence à des OID en SQL

```
1   INSERT INTO table1 (champs1, ..., clé_étrangère)
2     SELECT 'valeur1', ...,  REF(alias)
3     FROM table2 alias
4     WHERE clé_table2='valeur';
```

b) Héritage en relationnel-objet sous Oracle

Syntaxe

```
1  CREATE TYPE sur_type AS OBJECT (
2  ...
3  ) NOT FINAL;
4  /
5  CREATE TYPE sous_type UNDER sur_type (
6    Déclarations spécifiques ou surcharges
7  ) ;
8
```

Remarque : NOT FINAL

Pour être *héritable*, un type doit être déclaré avec la clause optionnelle `NOT FINAL`.

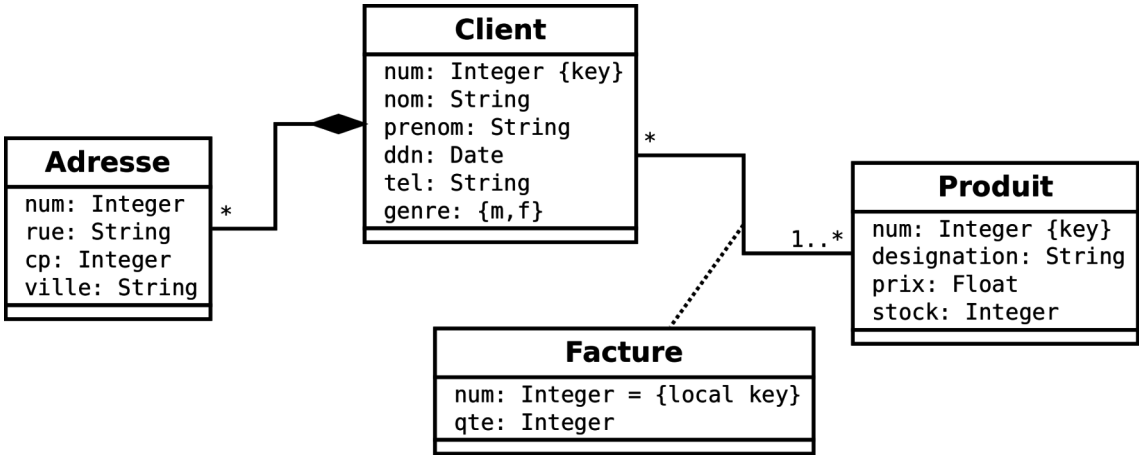
Exercices

II

A. MediaTek V

[1 h]

Facture n'est pas une table objet ici, elle n'est pas référencée




```
1  Adresse (#num:integer, #rue:string, #cp:integer, #ville:string, #client =>
   Client>)
2
3  Type TClient <num:integer, nom:string, prenom:string, ddn:date, tel:string,
   genre:{'m','f'}>
4  Client de TClient (#num)
5
6  Type TProduit <num:integer, designation:string, prix:float, stock:integer>
7  Produit de TProduit (#num)
8
9  Type TFacture <num:integer, client =>o Client, produit =>o Produit, qte:integer>
10 Facture de TFacture (#num, #client, #produit)
```

Question 1

[*Solution n°5 p 19*]

Réalisez l'implémentation SQL3 sous Oracle.

Alimenter une base RO

Une fois la base créée, insérer des données pour l'alimenter.

On négligera la table Adresse pour la suite de l'exercice (aucun de nos clients n'aura d'adresse).

Question 2

[Solution n°6 p 19]

Initialiser les tables client et produit avec les données de votre choix (au moins deux clients et deux produits).

Indice :

Effectuez l'insertion comme en relationnel.

Question 3

[Solution n°7 p 20]

Initialiser les factures avec les données de votre choix (deux factures de deux lignes chacune au moins).

Indices :

*Récupérer les **OID** pour insérer les références aux produits et clients au sein d'une procédure PL/SQL.*

Insertion de références par OID (INSERT)

Blocs PL/SQL : Procédure, fonction, bloc anonyme - p.27

```
1  DECLARE
2    oidclient1 REF TClient;
3    ...
4
5  BEGIN
6    SELECT REF(c) INTO oidclient1
7    FROM Client c
8    WHERE c.num=...;
9
10   ...
11
12   INSERT INTO Facture (num, client, produit, qte)
13   VALUES (1,oidclient1, ..., 3);
14
15   ...
16
17 END;
18 /
```

Questions par navigation d'OID

Vous allez à présent expérimenter la manipulation des OID pour **naviguer** d'enregistrement en enregistrement, sans utiliser de jointure.

Question 4

[Solution n°8 p 20]

Écrivez une requête permettant d'afficher la liste des factures existantes, avec le nom et le prénom du client.

Indice :

```
1  NUM CLIENT.NOM           CLIENT.PRENOM
2  --- -------------------- --------------------
3    1 Colomb               Christophe
4    1 Morin                Bernard
```

Question 5

[Solution n°9 p 20]

Écrivez une requête permettant d'afficher le montant total de chaque facture, en rappelant le numéro du client pour chaque facture.

Indice :

```
NUM CLI TOTAL
--- --- -----
  1   1 169.1
  1   2 319.3
```

Question 6

[*Solution n°10 p 21*]

Écrivez une requête permettant d'afficher pour chaque client (num et nom) les produits (num et désignation) qu'il a déjà acheté, avec la quantité correspondante.

Indice :

```
CLIENT.NUM CLIENT.NOM           PRODUIT.NUM PRODUIT.DESIGNATION  QTE
---------- -------------------- ----------- -------------------- ---
         1 Colomb                         1 The Matrix             3
         1 Colomb                         2 The Hobbit             2
         2 Morin                          1 The Matrix             1
         2 Morin                          2 The Hobbit             6
```

B. Des voitures et des hommes

[45 minutes]

Soit le diagramme de classe UML suivant :

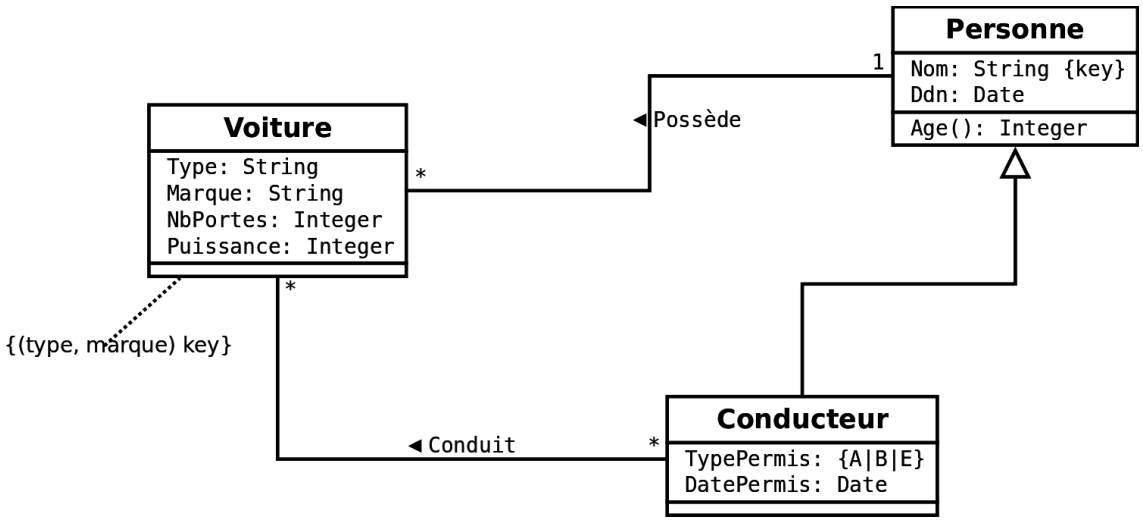


Image 1

Question 1

[*Solution n°11 p 21*]

A partir de ce modèle conceptuel établissez un modèle logique en relationnel-objet.

On utilisera des tables d'objets et les OID pour effectuer les références (en revanche, on n'utilisera pas le modèle imbriqué).

Question 2

[*Solution n°12 p 21*]

Proposer une implémentation sous Oracle de votre modèle logique (sans implémenter les méthodes et sans utiliser l'héritage de type).

Question 3

[Solution n°13 p 22]

Implémenter la méthode Age() pour Personne et pour Conducteur.

Indice :

```
1 trunc(months_between(SYSDATE, ddn)/12
```

Question complémentaire (héritage de type)

L'amélioration proposée ci-après évite notamment la double déclaration de la méthode `age`.

Question 4

[Solution n°14 p 22]

Proposer une solution mobilisant l'héritage de type afin de factoriser la déclaration des attributs `nom` et `ddn` et la méthode `age`.

C. Des voitures et des hommes de collection

[45 minutes]

Soit le diagramme de classe UML suivant :

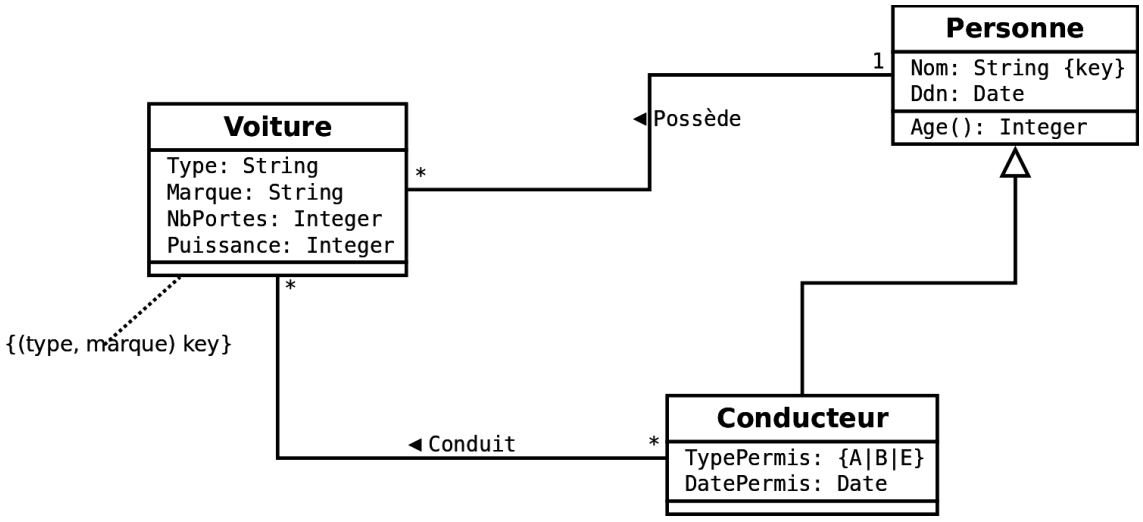


Image 2

Question 1

[Solution n°15 p 23]

A partir de ce modèle conceptuel établissez un modèle logique en relationnel-objet.

On utilisera des tables d'objets et les OID pour effectuer les références, ainsi qu'une table imbriquée pour gérer l'association N:M comme une collection de référence à des OID.

On privilégiera la table Voiture, qui sera au centre des requêtes posées à la BD.

Question 2

[Solution n°16 p 23]

Proposer une implémentation sous Oracle de votre modèle logique (sans implémenter les méthodes et sans utiliser l'héritage de type).

Question 3

[Solution n°17 p 24]

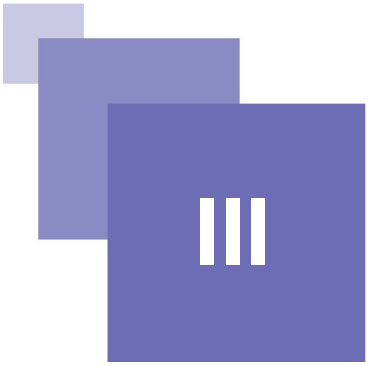
Insérer une voiture, un propriétaire et deux conducteurs dans le base de données.

Question 4

[*Solution n°18 p 24*]

Afficher toutes les voitures avec les personnes qui les possèdent et les conduisent.

Devoirs



A. Zoologie

[45 minutes]

Un zoologue souhaite réaliser une base de données pour l'étude des chaînes alimentaires.

Il souhaite étudier deux populations d'animaux : les carnivores et les herbivores, sachant que les carnivores ne mangent **que** des herbivores et que les herbivores, évidemment, ne mangent pas d'animaux. La base gère donc des espèces d'animaux, identifiées par leur nom (Lion, Biches, etc.). Les carnivores et les herbivores sont étudiés selon deux paramètres communs : taille moyenne et poids moyen.

Pour les herbivores, on étudie en plus le poids moyen de végétaux mangé par jour. Bien entendu le zoologue souhaite connaître également le poids moyen mangé par jour pour chaque espèce de carnivore et pour chaque espèce d'herbivore qu'il mange. Il souhaite enfin pouvoir obtenir très facilement le poids total mangé par un carnivore par jour (donc en sommant le poids de chaque herbivore mangé par jour). On fera apparaître cette opération sur le modèle.

Question 1

Réalisez le modèle conceptuel du problème posé.

Question 2

Réalisez le modèle logique **relationnel-objet** du problème (on utilisera les OID pour toutes les tables référencés).

On ajoutera la méthode *poidsMangéParJourTotal()* à la table d'objets *Carnivore*.

Question 3

Implémentez le modèle relationnel-objet en SQL3 sous Oracle (sans implémenter de méthode).

Question 4

Implémentez la méthode permettant de calculer le poids total d'herbivore mangé par un carnivore par jour.

Question 5

Écrivez une requête qui permet de retourner tous les carnivores, triés par leur poids, avec le poids total d'herbivore qu'ils mangent par jour.

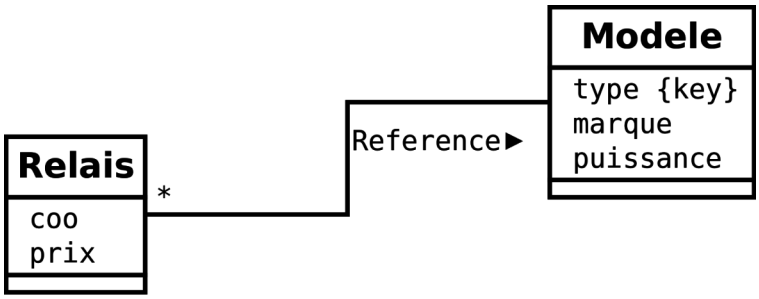
Question 6

Proposez et implémentez une solution d'optimisation pour améliorer les performances de la requête précédente. Justifiez votre choix et expliquez, éventuellement à l'aide d'un petit schéma ou d'un exemple, pourquoi l'exécution sera améliorée.

Solution des exercices

> Solution n°1 *(exercice p. 6)*

MCD



MLD

```
type ModeleT <type:varchar, marque:varchar, puissance:entier>
type RelaisT <id:entier, coo:chaine, prix:decimal, modele =>o Modele>

table Modele of ModeleT (#type)
table Relais of RelaisT (#id)
```

SQL

```
CREATE TYPE ModeleT AS OBJECT (
type varchar(25),
marque varchar(25),
puissance number
);
/
CREATE TABLE Modele OF ModeleT (
PRIMARY KEY (type)
);

CREATE or replace TYPE RelaisT AS OBJECT (
id number,
coo varchar(11),
prix number,
modele REF ModeleT
);
/
CREATE TABLE Relais OF RelaisT (
PRIMARY KEY (id),
modele NOT NULL,
SCOPE FOR (modele) IS Modele
```

```
22 );
```

> Solution n°2 *(exercice p. 6)*

```
1  DECLARE
2  x1 REF ModeleT;
3
4  BEGIN
5
6  -- Insertion du modèle X1
7  INSERT INTO Modele VALUES ('X1', 'SuperWif', 25);
8
9  -- Récupération de l'OID de X
10 SELECT REF(m) INTO x1
11 FROM Modele m
12 WHERE type='X1';
13
14 -- Insertion des relais
15 INSERT INTO Relais VALUES (1, '48.853,2.35',100,x1);
16 INSERT INTO Relais VALUES (2, '48.978,3.01',100,x1);
17
18 END;
```

> Solution n°3 *(exercice p. 6)*

```
1  SELECT r.modele.puissance
2  FROM Relais r
3  WHERE r.id=1
```

```
1  SELECT AVG(r.prix)
2  FROM Relais r
3  GROUP BY r.modele.type
```

> Solution n°4 *(exercice p. 8)*

```
1  CREATE TYPE FilmT AS OBJECT (
2  isan:char(33),
3  titre:varchar2,
4  debut:date,
5  fin:date,
6  MEMBER FUNCTION dureeTournage RETURN integer
7  );
8  /
9
10 CREATE TYPE BODY FilmT
11 IS
12 MEMBER FUNCTION dureeTournage RETURN integer)
13 IS
14 BEGIN
15   RETURN duree(SELF.debut,SELF.fin);
16 END;
17 END;
18 /
19
20 CREATE TABLE Film AS FilmT (
21 PRIMARY KEY (isan));
```

> Solution n°5 *(exercice p. 10)*

Client

```
1  CREATE TYPE TClient AS OBJECT (
2    num NUMBER(4),
3    nom VARCHAR(20),
4    prenom VARCHAR(20),
5    ddn DATE,
6    tel VARCHAR(10),
7    genre CHAR(1)
8  );
9  /
10 CREATE TABLE Client OF TClient(
11   PRIMARY KEY (num),
12   CHECK (genre IN ('m', 'f'))
13 );
14 CREATE TABLE Adresse (
15  num NUMBER(4),
16  rue VARCHAR(255),
17  cp NUMBER(5),
18   ville VARCHAR(255),
19   client NUMBER(4) REFERENCES Client (num),
20   PRIMARY KEY (num, rue, cp, ville, client)
21 );
```

Produit

```
1  CREATE TYPE TProduit AS OBJECT (
2    num NUMBER(10),
3    designation VARCHAR(128),
4    prix NUMBER(5,2),
5    stock NUMBER(4)
6  );
7  /
8  CREATE TABLE Produit of TProduit (PRIMARY KEY (num));
```

Facture

```
1  CREATE TYPE TFacture AS OBJECT (
2    num NUMBER(10),
3    client REF TClient,
4    produit REF TProduit,
5    qte NUMBER(4)
6  );
7  /
8  CREATE TABLE Facture OF TFacture (
9    SCOPE FOR (client) IS Client,
10   SCOPE FOR (produit) IS Produit
11 );
```

Rappel

Sous Oracle on ne pourra créer de contrainte UNIQUE incluant un OID, donc ici on ne peut pas spécifier la contrainte (#num, #client, #produit).

> Solution n°6 *(exercice p. 11)*

```
1  INSERT INTO Client (num, nom, prenom, ddn, tel, genre)
2  VALUES (1, 'Colomb', 'Christophe', to_date('01091451', 'DDMMYYYY'),
   '0812456589' , 'm');
3
```

```
4  INSERT INTO Client (num, nom, prenom, ddn, tel, genre)
5  VALUES (2, 'Morin', 'Bernard', to_date('27051931', 'DDMMYYYY'), '0126483621' ,
   'm');
6
7  INSERT INTO Produit (num, designation, prix, stock)
8  VALUES (1, 'The Matrix', 23.5, 100) ;
9
10 INSERT INTO Produit (num, designation, prix, stock)
11 VALUES (2, 'The Hobbit', 49.3, 100) ;
```

> Solution n°7 *(exercice p. 11)*

```
1  DECLARE
2    oidclient1 REF TClient;
3    oidproduit1 REF TProduit;
4    oidclient2 REF TClient;
5    oidproduit2 REF TProduit;
6
7  BEGIN
8    SELECT REF(c) INTO oidclient1
9    FROM Client c
10   WHERE c.num=1;
11
12   SELECT REF(c) INTO oidclient2
13   FROM Client c
14   WHERE c.num=2;
15
16   SELECT REF(p) INTO oidproduit1
17   FROM Produit p
18   WHERE p.num=1;
19
20   SELECT REF(p) INTO oidproduit2
21   FROM Produit p
22   WHERE p.num=2;
23
24   INSERT INTO Facture (num, client, produit, qte)
25   VALUES (1, oidclient1, oidproduit1, 3);
26
27   INSERT INTO Facture (num, client, produit, qte)
28   VALUES (1, oidclient1, oidproduit2, 2);
29
30   INSERT INTO Facture (num, client, produit, qte)
31   VALUES (1, oidclient2, oidproduit2, 6);
32
33   INSERT INTO Facture (num, client, produit, qte)
34   VALUES (1, oidclient2, oidproduit1, 1);
35
36 END;
37 /
```

> Solution n°8 *(exercice p. 11)*

```
1  COLUMN num FORMAT A3
2  SELECT f.num, f.client.nom, f.client.prenom
3  FROM Facture f
4  GROUP BY f.num, f.client.nom, f.client.prenom;
```

> Solution n°9 *(exercice p. 11)*

```
1  COLUMN CLI FORMAT A3
2  SELECT f.num, f.client.num AS cli, SUM(f.qte * f.produit.prix) AS total
3  FROM Facture f
4  GROUP BY f.num, f.client.num
```

```
5  ORDER BY f.num, f.client.num
```

> Solution n°10 *(exercice p. 12)*

```
1  COLUMN PRODUIT.NUM FORMAT A11
2  COLUMN PRODUIT.DESIGNATION FORMAT A20
3  SELECT f.client.num, f.client.nom, f.produit.num, f.produit.designation,
   SUM(f.qte) AS qte
4  FROM Facture f
5  GROUP BY f.client.num, f.client.nom, f.produit.num, f.produit.designation
6  ORDER BY f.client.num, f.client.nom, f.produit.num, f.produit.designation
```

> Solution n°11 *(exercice p. 12)*

```
1  Type Personne : <
2    nom:string,
3    ddn:date,
4    =age():entier
5  >
6  tPersonne de Personne (#nom)
7
8  Type Conducteur : <
9    nom:string,
10   ddn:date,
11   typePermis:{A,B,E},
12   datePermis:date
13   =age():entier
14 >
15 tConducteur de Conducteur (#nom)
16
17 Type Voiture : <
18   type:string,
19   marque:string,
20   nbPortes:integer,
21   puissance:integer,
22   fkPropriétaire =>o tPersonne,
23 >
24 tVoiture de Voiture (#type, #marque)
25
26 tConduit (voiture =>o tVoiture, conducteur =>o tConducteur)
```

Rappel

Il s'agit d'un cas simple, on adopte un héritage par les classes filles.

- Héritage non complet
- Classe mère non abstraite
- Sans association X:N vers la classe mère

Cas simples - p.29

Cas problématiques - p.31

> Solution n°12 *(exercice p. 12)*

```
1  CREATE OR REPLACE TYPE Personne AS OBJECT (
2    nom varchar(20),
3    ddn date,
4    MEMBER FUNCTION age RETURN number
5  );
6  /
7  CREATE TABLE tPersonne OF Personne (
8    PRIMARY KEY (nom)
9  );
```

```
10
11  CREATE OR REPLACE TYPE Conducteur AS OBJECT(
12    nom varchar(20),
13    ddn date,
14    typePermis char(1),
15    datePermis date,
16    MEMBER FUNCTION age RETURN number
17  );
18  /
19  CREATE TABLE tConducteur OF Conducteur (
20    PRIMARY KEY (nom),
21    CHECK (typePermis IN ('A','B','E'))
22  );
23
24  CREATE OR REPLACE TYPE Voiture AS OBJECT (
25    type varchar(20),
26    marque varchar(20),
27    nbPortes number,
28    puissance number,
29    proprietaire REF Personne
30  );
31  /
32  CREATE TABLE tVoiture OF Voiture (
33    PRIMARY KEY (type, marque),
34    SCOPE FOR (proprietaire) IS tPersonne
35  );
36
37  CREATE TABLE tConduit (
38    voiture REF Voiture,
39    conducteur REF Conducteur,
40    SCOPE FOR (voiture) IS tVoiture,
41    SCOPE FOR (conducteur) IS tConducteur
42  );
```

> Solution n°13 *(exercice p. 13)*

```
1   CREATE OR REPLACE TYPE BODY Personne
2   IS
3     MEMBER FUNCTION age RETURN number
4     IS
5     BEGIN
6       RETURN trunc(months_between(SYSDATE, ddn)/12);
7     END;
8   END;
9   /
10  CREATE OR REPLACE TYPE BODY Conducteur
11  IS
12    MEMBER FUNCTION age RETURN number
13    IS
14    BEGIN
15      RETURN trunc(months_between(SYSDATE, ddn)/12);
16    END;
17  END;
18  /
```

Remarque

On remarque la duplication de la méthode `age`.

On peut éviter cela en utilisant l'héritage de type.

> Solution n°14 *(exercice p. 13)*

```
1   CREATE OR REPLACE TYPE Personne AS OBJECT (
2     nom varchar(20),
```

```
  ddn date,
  MEMBER FUNCTION age RETURN number
) NOT FINAL;
/
CREATE OR REPLACE TYPE Conducteur UNDER Personne (
  typePermis char(1),
  datePermis date
);
```

> Solution n°15 *(exercice p. 13)*

```
Type Personne : <
  nom:string,
  ddn:date,
  =age():entier
>
tPersonne de Personne (#nom)

Type Conducteur : <
  nom:string,
  ddn:date,
  typePermis:{A,B,E},
  datePermis:date
  =age():entier
>
tConducteur de Conducteur (#nom)

Type RefConducteur : <refConducteur =>o tConducteur>
Type ListeRefConducteur : collection de <RefConducteur>

Type Voiture : <
  type:string,
  marque:string,
  nbPortes:integer,
  puissance:integer,
  refPropriétaire =>o tPersonne,
  refConducteurs:ListeRefConducteur
>
tVoiture de Voiture (#type, #marque)
```

> Solution n°16 *(exercice p. 13)*

```
CREATE OR REPLACE TYPE Personne AS OBJECT (
  nom varchar(20),
  ddn date,
  MEMBER FUNCTION age RETURN number
);
/
CREATE TABLE tPersonne OF Personne (
  PRIMARY KEY (nom)
);

CREATE OR REPLACE TYPE Conducteur AS OBJECT(
  nom varchar(20),
  ddn date,
  typePermis char(1),
  datePermis date,
  MEMBER FUNCTION age RETURN number
);
/
CREATE TABLE tConducteur OF Conducteur (
  PRIMARY KEY (nom),
  CHECK (typePermis IN ('A','B','E'))
);

```

```
24  CREATE OR REPLACE TYPE RefConducteur AS OBJECT (refConducteur REF Conducteur);
25  /
26  CREATE OR REPLACE TYPE ListeRefConducteur AS TABLE OF RefConducteur;
27  /
28  CREATE OR REPLACE TYPE Voiture AS OBJECT (
29    type varchar(20),
30    marque varchar(20),
31    nbPortes number,
32    puissance number,
33    refProprietaire REF Personne,
34    refConducteurs ListeRefConducteur
35  );
36  /
37  CREATE TABLE tVoiture OF Voiture (
38    PRIMARY KEY (type, marque),
39    SCOPE FOR (refProprietaire) IS tPersonne
40  )
41  NESTED TABLE refConducteurs STORE AS ntRefConducteurs;
42
```

Remarque

On notera que le choix d'implémentation de *fkConducteurs* en collection de références à des OID nous empêche dans l'implémentation sous Oracle de définir le *SCOPE FOR* et donc de contraindre l'intégrité référentielle à la table *tConducteur*. Dans le cas présent ce n'est pas préjudiciable dans la mesure où seule la table *tConducteur* contient des objets *Conducteur*.

> Solution n°17 *(exercice p. 13)*

```
1   DECLARE
2
3   c1 REF Conducteur;
4   c2 REF Conducteur;
5   p1 REF Personne;
6
7   BEGIN
8
9   INSERT INTO tPersonne (nom) VALUES ('Al');
10
11  SELECT REF(p) INTO p1
12  FROM tPersonne p
13  WHERE nom='Al';
14
15  INSERT INTO tConducteur (nom, typePermis) VALUES ('Bob', 'A');
16
17  SELECT REF(c) INTO c1
18  FROM tConducteur c
19  WHERE nom='Bob';
20
21  INSERT INTO tConducteur (nom, typePermis) VALUES ('Charlie', 'B');
22
23  SELECT REF(c) INTO c2
24  FROM tConducteur c
25  WHERE nom='Charlie';
26
27  INSERT INTO tVoiture (type, marque, refProprietaire, refConducteurs)
28  VALUES ('Superstar', 'S12', p1,
      ListeRefConducteur(RefConducteur(c1),RefConducteur(c2)));
29
30  END;
```

> Solution n°18 *(exercice p. 14)*

```
1   SELECT v.type, v.marque, v.refProprietaire.nom, c.refConducteur.nom
```

```
2  FROM tVoiture v, TABLE(v.refConducteurs) c;
```

Index

Contenus annexes

- Création de type en SQL3 sous Oracle (extension au LDD)

Syntaxe : Déclaration de type

```
1  CREATE TYPE nom_type AS OBJECT (
2    nom_attribut1 type_attribut1,
3    ...
4  );
5  /
6
```

Exemple : Création de tables d'objets (enregistrements avec OID)

```
1  CREATE TABLE t OF nom_type (
2    ...
3  )
```

Exemple : Usage des types dans les tables (modèle imbriqué)

```
1  CREATE TABLE t (
2    ...
3    nom_attribut nom_type,
4    ...
5  )
```

Complément

Héritage et réutilisation de types
Méthodes de table d'objets

- Blocs PL/SQL : Procédure, fonction, bloc anonyme

Syntaxe : Procédure

```
1  CREATE OR REPLACE PROCEDURE nom_proc
2  IS
3    ...
4  BEGIN
5    ...
6  [EXCEPTION]
7    ...
8  END ;
```

Exemple : Procédure

```
1  CREATE OR REPLACE PROCEDURE pHello (who VARCHAR2)
```

```
IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello ' || who);
END;
/
```

Syntaxe : Fonction

```
CREATE OR REPLACE FUNCTION nom_func
  RETURN type_retourné
IS
  ...
BEGIN
  ...
  RETURN valeur;
[EXCEPTION]
  ...
END ;
```

Exemple : Fonction

```
CREATE OR REPLACE FUNCTION fDateDuJour RETURN date
IS
  vDate date;
BEGIN
  SELECT SYSDATE INTO vDate FROM DUAL;
  RETURN vDate;
END;
/
```

Attention

Le type de retourné par une fonction ne doit pas spécifier de taille :

- **`RETURN varchar`**
- **et non `RETURN varchar(10)`**

***http://docs.oracle.com/cd/B13789_01/server.101/b10759/statements_5009.htm*[2]**

Syntaxe : Anonyme

```
[DECLARE]
  ...
BEGIN
  ...
[EXCEPTION]
  ...
END ;
```

Exemple : Script anonyme

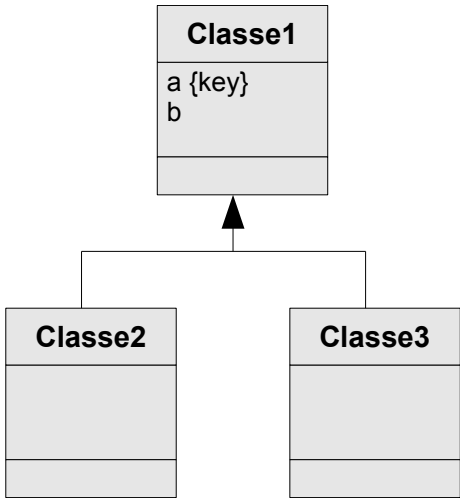
```
SET SERVEROUTPUT ON;
BEGIN
  pHello('World');
  DBMS_OUTPUT.PUT_LINE(fDateDuJour);
END;
/
```

2 - http://docs.oracle.com/cd/B13789_01/server.101/b10759/statements_5009.htm

- Cas simples

Méthode : Héritage complet

Dans ce cas, choisir un **héritage par la classe mère**.

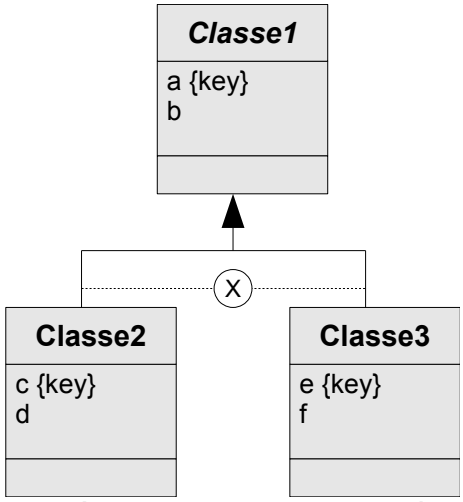


Graphique 1 Héritage complet

1. Si la classe mère est abstraite :
   `Classe1(#a,b,t:{2,3})`
2. Si la classe mère n'est pas abstraite :
   `Classe1(#a,b,t:{1,2,3})`

Méthode : Héritage non complet avec classe mère abstraite et sans association

Dans ce cas, choisir un **héritage par les classes filles**.

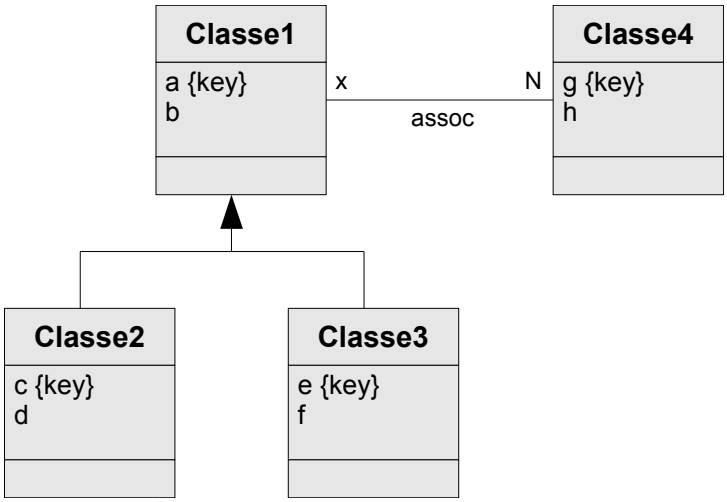


Graphique 2 Héritage exclusif (classe mère abstraite)

`Classe2(#a,b,c,d) avec c KEY`
`Classe3(#a,b,e,f) avec e KEY`

Méthode : Héritage presque complet

L'héritage presque complet peut être géré comme l'héritage complet, **par la classe mère**, surtout si les classes filles ne possèdent pas de clé propre.

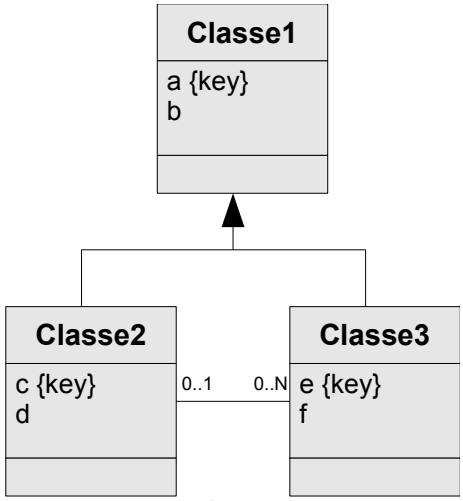


Graphique 3 Héritage avec association x:N sur la classe mère

```
Classe1(#a,b,c,d,e,f,t:{1,2,3})
Classe4(#g,h,fka=>Classe1)
```

Méthode : Héritage non complet avec classe mère non abstraite et sans association

Dans ce cas, choisir un **héritage par les classes filles**.



Graphique 4 Héritage non complet

```
Classe1(#a,b)
Classe2(#a,b,c,d) avec c KEY
Classe3(#a,b,e,f,fka=>Classe2) avec e KEY
```

- Cas problématiques

Attention : Héritage par les classes filles avec association M:N ou 1:N sur la classe mère

Héritage avec association x:N sur la classe mère

On traite le cas x=1 (1:N) avec Classe1 abstraite (mais le cas M:N et/ou classe non abstraite ne change rien au problème) :

`Classe2(#a,b,c,d) avec c KEY`

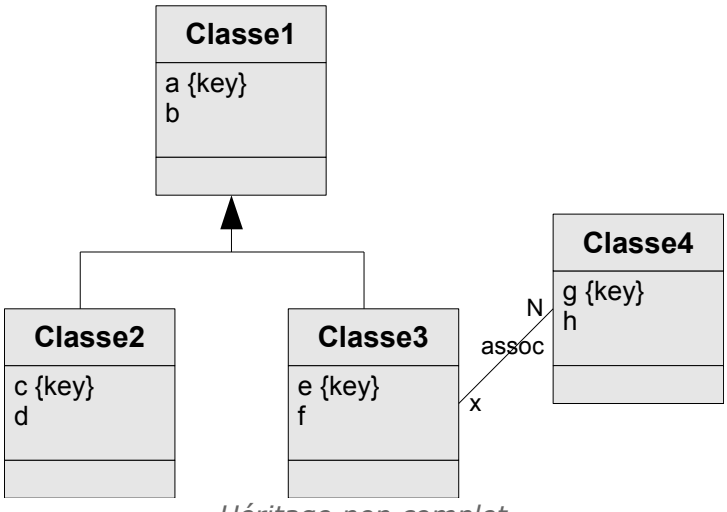
`Classe3(#a,b,e,f) avec e KEY`

`Classe4(#g,h,fka=>Classe2, fkb=>Classe3)`

Contrainte : `fka OR fkb`

La seule solution, peu élégante, consiste à ajouter autant de clés étrangères que de classes filles et à gérer le fait que ces clés ne peuvent pas être co-valuées.

Attention : Héritage non complet par la classe mère (association M:N ou 1:N sur une classe fille)



Héritage non complet

On traite le cas x=1 (1:N) (mais le cas M:N ne change rien au problème) :

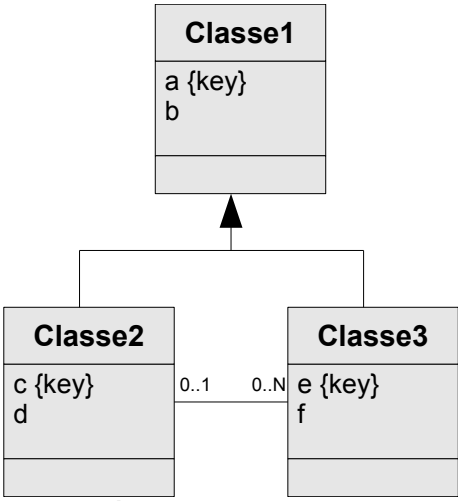
`Classe1(#a,b,c,d,e,f,t:{1,2,3})`

`Classe4(#g,h,fka=>Classe1)`

Contraintes : `Classe4.fka ne référence que des enregistrements tels que Classe1.t=3`

On est obligé d'ajouter une contrainte pour limiter la portée de la clé étrangère de Classe4 ; on est sorti ici de ce que l'on sait faire de façon simple en relationnel.

Attention : Héritage non complet par la classe mère (association entre classes filles)



Héritage non complet

`Classe1(#a,b,c,d,e,f,fka=>Classe1,t:{1,2,3})`

Contraintes : `fka ne référence que des enregistrements tels que t=2 ; si fka alors t=3`

Dans ce cas la solution est encore plus problématique, elle permettra en l'état des associations entre Classe1 et Classe3, et même entre Classe3 et Classe3, on est très loin de la modélisation conceptuelle initiale.

Conseil

Afin de déterminer si un héritage est presque complet ou non, il faut donc surtout regarder les associations, ce sont elles qui poseront le plus de problème un fois en relationnel (à cause de l'intégrité référentielle).

Complément : Héritage non exclusif

L'héritage non exclusif ne doit pas être traité par les classes filles, sous peine d'introduire de la redondance.

Complément : Héritage multiple

L'héritage multiple sera généralement mieux géré avec un héritage par référence.

Complément

Héritage complet - p.33
Classes abstraites - p.33

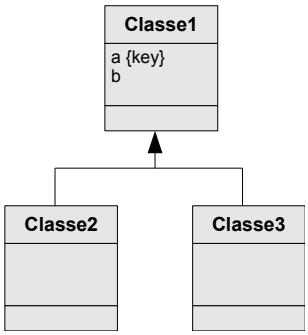
Complément : Les cas problématiques obligent à ajouter des contraintes

Contraintes en UML - p.34
Liste des contraintes en R - p.35

- Héritage complet

Définition : Héritage complet et presque complet

Un héritage est complet si ses classes filles n'ont aucune caractéristiques (attributs, méthodes, associations) propres.



Graphique 5 Héritage complet

Un héritage est presque complet si les classes filles ont des méthodes propres, quelques attributs propres, et **aucune association propre**.

- Classe abstraite

Définition : Classe abstraite

Une classe abstraite est une classe non instanciable. Elle exprime donc une généralisation abstraite, qui ne correspond à aucun objet existant du monde.

Attention : Classe abstraite et héritage

Une classe abstraite est toujours héritée. En effet sa fonction étant de généraliser, elle n'a de sens que si des classes en héritent. Une classe abstraite peut être héritée par d'autres classes abstraites, mais en fin de chaîne des classes non abstraites doivent être présentes pour que la généralisation ait un sens.

Syntaxe : Notation d'une classe abstraite en UML

On note les classes abstraites en italique.

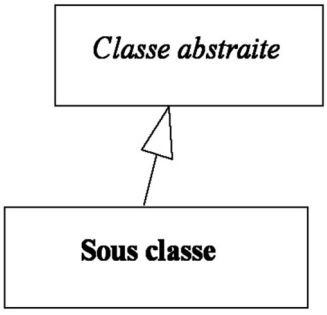


Image 3 Notation d'une classe abstraite en UML

Exemple : Exemple de classes abstraites

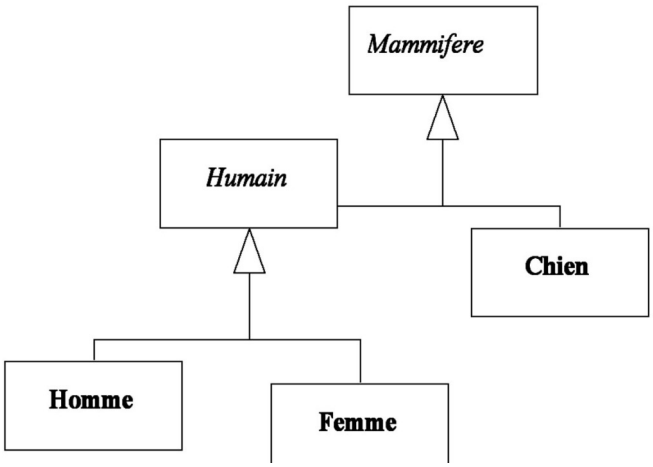


Image 4 Des chiens et des hommes

Dans la représentation précédente on a posé que les hommes, les femmes et les chiens étaient des objets instanciables, généralisés respectivement par les classes mammifère et humain, et mammifère.

Selon cette représentation, il ne peut donc exister de mammifères qui ne soient ni des hommes, ni des femmes ni des chiens, ni d'humains qui ne soient ni des hommes ni des femmes.

- Contraintes

Ajout de contraintes dynamiques sur le diagramme de classe

Il est possible en UML d'exprimer des contraintes dynamiques sur le diagramme de classe, par annotation de ce dernier.

Syntaxe : Notation de contraintes

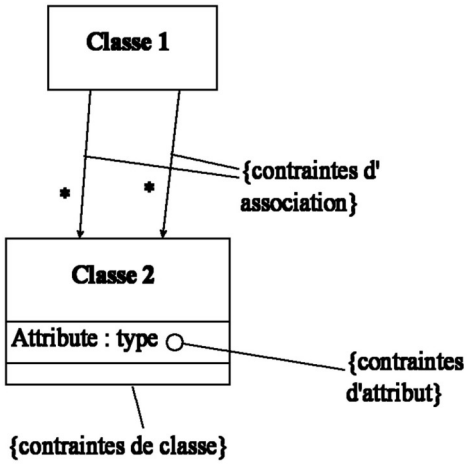


Image 5 Notation contraintes en UML

Exemple : Exemple de contraintes

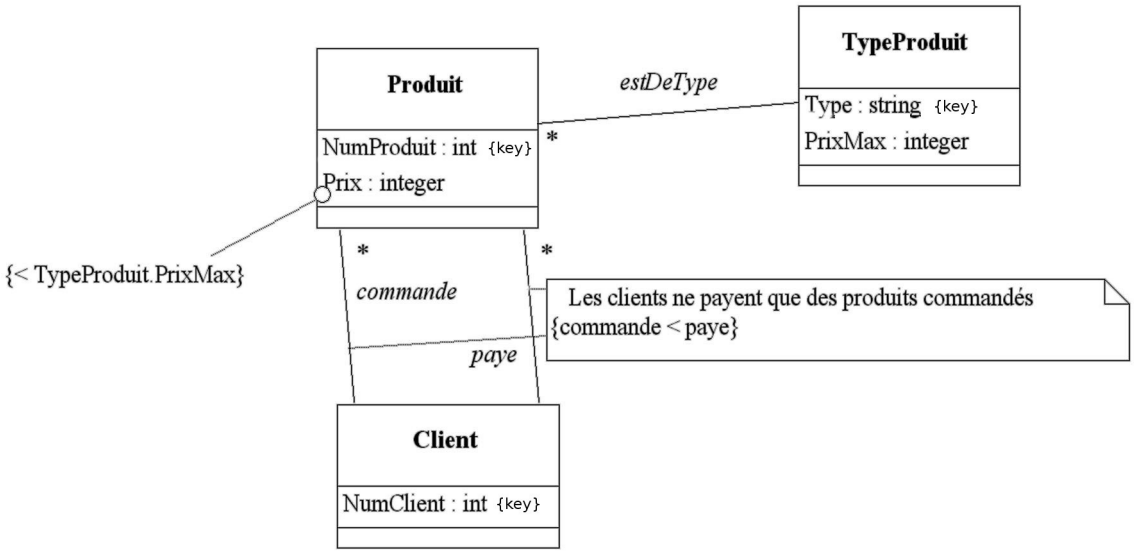


Image 6 Commandes

Méthode : Expressions formelles ou texte libre ?

Il est possible d'exprimer les contraintes en texte libre ou bien en utilisant des expressions formelles.

On privilégiera la solution qui offre le meilleur compromis entre facilité d'interprétation et non ambiguïté. La combinaison des deux est également possible si nécessaire.

Méthode : Quelles contraintes exprimer ?

En pratique il existe souvent de très nombreuses contraintes, dont certaines sont évidentes, ou encore secondaires. Or l'expression de toutes ces contraintes sur un diagramme UML conduirait à diminuer considérablement la lisibilité du schéma sans apporter d'information nécessaire à la compréhension. En conséquence on ne représentera sur le diagramme que les contraintes les plus essentielles.

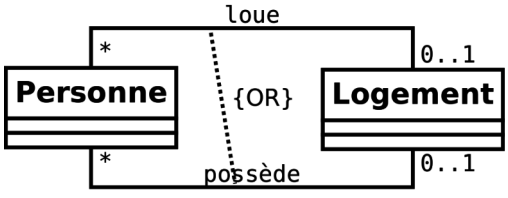
Notons que lors de l'implémentation toutes les contraintes devront bien entendu être exprimées. Si l'on souhaite préparer ce travail, il est conseillé, lors de la modélisation logique, de recenser de façon exhaustive dans un tableau toutes les contraintes à implémenter.

- Liste des contraintes

Méthode : Contraintes exprimées sur le MCD

Les contraintes exprimées au niveau conceptuel doivent être traduite au niveau relationnel.

Si les contraintes ne sont pas trop nombreuses, on peut commenter chaque relation directement lors de la formalisation du MLD.



Exemple de contrainte OR entre deux associations

```
1  Personne (...)
2  Logement (... loue=>Personne, possède=>Personne) avec (loue OR possède)
```

Rappel : UNIQUE, NOT NULL

On pensera à exprimer les clés candidates KEY (ou UNIQUE NOT NULL), les attributs UNIQUE et NOT NULL.

```
1  Classe1 (#pk, k, u, a) avec k KEY, u UNIQUE, a NOT NULL
```

Méthode

Si l'expression des contraintes nuit à la lisibilité, on les reportera dans un document annexe qui accompagnera le modèle logique.

Conseil : Extension des contraintes exprimées

On s'attachera lors de la modélisation logique à exprimer l'ensemble des contraintes dynamiques pesant sur le modèle, même celles qui ont été considérées comme secondaires ou évidentes lors de la modélisation conceptuelle.